Towards explaining rebuttals in security arguments

Conference or Workshop Item

Towards Explaining Rebuttals in Security Arguments

Yijun YU [a,1], Paul PIWEK [a] and Thein Than TUN [a] and Bashar NUSEIBEH [a,b]
[a] *Department of Computing and Communications, The Open University, UK*
[b] *Lero, The Irish Software Engineering Research Centre, University of Limerick, Ireland*

Abstract. The satisfaction of software security requirements can be argued using supporting facts and domain assumptions. Sometimes, these facts or assumptions may be questioned, as more knowledge about vulnerabilities becomes available. This results in rebuttals that can be derived from the new information. In this paper, we outline an extension of our `OpenArgue` tool with an explanation facility that makes a rebuttal more transparent by showing, step by step, why the original security argument does not hold. We achieve this by using the output of the ALLIGATOR theorem prover, which constructs explicit and checkable proof objects. We illustrate the feasibility of this approach by applying it to an existing case study of a PIN entry device which involves a security argument that has been rebutted. The output of the prover enables us to unpack the logical reasoning behind the rebuttal at a much greater level of detail. This promises to be useful for argument explanation.

Keywords. Security requirements, Proof systems, Formal arguments

1. Introduction

Security threats and vulnerabilities in software systems are evolving rapidly. Not long after the 'heartbleed' vulnerability of the `OpenSSL` implementation was reported,[2] a more severe 'shellshock' vulnerability was found to have impacted on any `bash`-based unix distributions,[3] followed by the discovery of the 'Poodle' bug in the `SSLv3` used by most web browsers.[4] Security mechanisms that were once trusted are no longer so. In hindsight, it can be 'easy' to find facts against previously established *trust assumptions*, or arguments in support of security claims. Of course, after fixing the vulnerabilities, a software vendor may (at least temporarily) restore faith in the security of a system by updating security arguments about mitigating with counter-measures deployed.

A logical analysis of the satisfaction of security requirements leads to an iterative argumentation process. An initial claim of security goes through rebuttals and mitigations, because facts become debatable due to the lack of complete knowledge about the design of software systems, the model of potential attacks, and the organisational and resource

[1] Email: yijun.yu@open.ac.uk
[2] http://www.bbc.co.uk/news/technology-26969629
[3] http://www.bbc.co.uk/news/technology-29375636
[4] http://www.bbc.co.uk/news/technology-29627887

constraints. Debatable assumptions or facts are referred to as *trust assumptions*, because retaining them requires a leap of faith/trust [5]. The "risk assessment in security argumentation" (RISA) approach [4] was proposed in order to expose these trust assumptions and assess the security risks that were not included in the initial analysis of security requirements.

Even though questionable trust assumptions are often found after successful attacks, the detailed reasons that lead to the denial of security requirements are usually not made explicit. Without such understanding of the reasons, it is likely that similar mistakes will be repeated in the future. In this paper, we investigate, using a case study, the feasibility of exploiting the output of a theorem prover with explicit proof objects to explain rebuttals of security arguments.

2. PIN Entry Device example

Before introducing the proof system, we first illustrate the application of security arguments using a realistic PIN Entry Device (PED) example. The PED is a type of device widely-deployed and used by consumers to pay for goods with debit or credit smartcards at the Points-Of-Sale (POS). When using the device, cardholders typically insert their cards, issued by a financial institution, into a card-reader interface, enter the PIN using the PED's keypad, and confirm the transaction value via a display on the PED itself. Then smartcard-based systems are expected to authenticate cardholders via the PIN and verify the card details against a public-key certificate before transactions can be completed. These certificates are usually stored on the chip but they can also be stored on the magnetic strip for compatibility with card-readers that have not adopted this technology.

Most PEDs used in Europe implement the EMV (EuroPay, MasterCard and Visa) protocol in the process of authentication and authorization of payment transactions. This protocol drives the communication at the PED-card interface and the PED-bank interface. The protocol by design allows only asymmetrically encrypted transmission of the PIN across these interfaces. However, many card issuers in Europe make the conscious decision to adopt a low-cost EMV option in their smartcards which researchers [3] have found to the be vulnerable, since it can be triggered to transmit an unencrypted PIN via the interface between the PED and card.

To illustrate the problem associated with this example, we constructed an incrementally updated argument to mimic the process of rebuttal and mitigation of security requirements, see Figure 1. In this argumentation diagram, nodes represent the claims, with factual ground and warrants nested inside. The dotted arrow from a node A to node B indicates rebuttal of the earlier claim in node A using the facts nested under node B. The trust assumptions nested inside the nodes are revealed when facts nested in the rebuttal nodes are found to be inconsistent with them, as indicated by the arrow between the facts internal to the nodes.

Using the `OpenArgue` tool [11],[5] the structured argumentation diagram can be automatically transformed into a logic formula in Event Calculus. If this formula is inconsistent, rebuttal relationships can be inferred. However, the existence proof of the rebuttal relationship between two collections of facts doesn't allow us to explain *why* inconsistency with the trust assumptions has led to the denial of the initial claim.

[5] http://sead1.open.ac.uk/risa

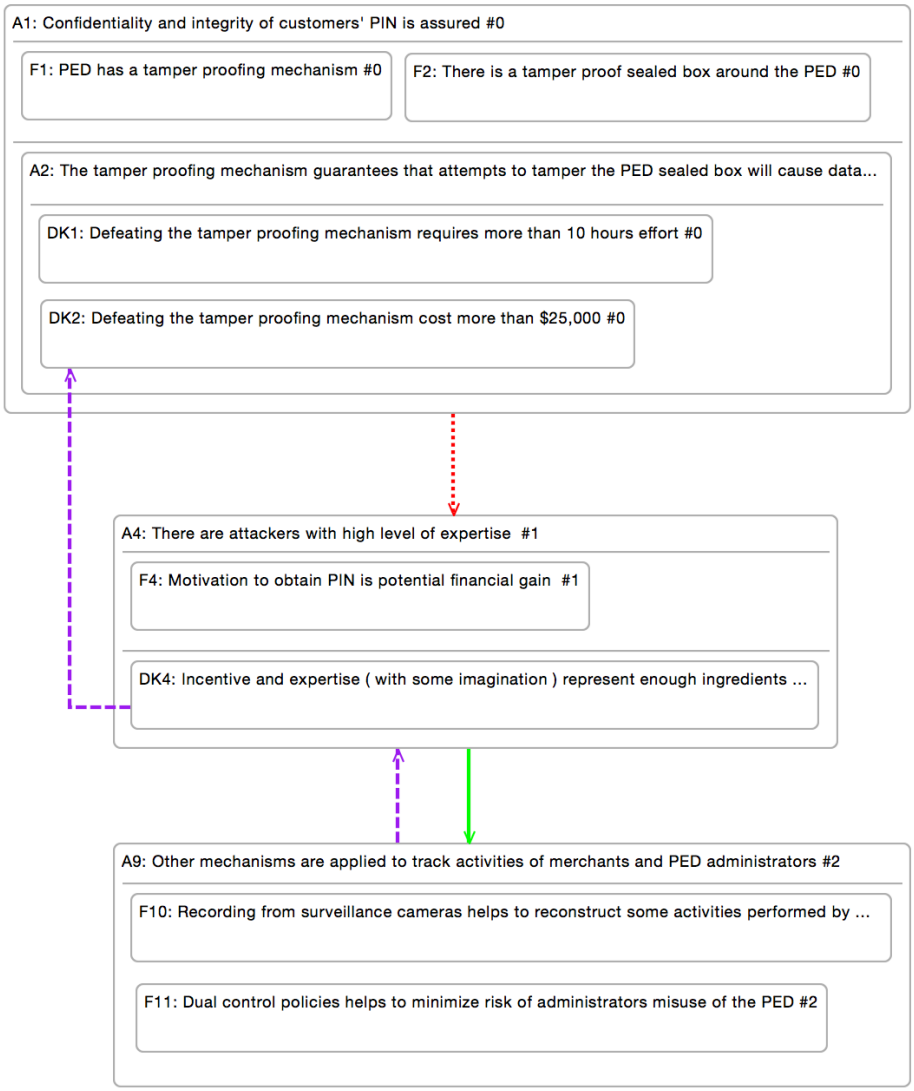

Figure 1. Some rebuttals and mitigations about the security claim of the PED system: the upwards arrows (e.g., DK4 $\rightarrow$ DK2, A9 $\rightarrow$ A4) indicate from which facts and domain knowledge the trust assumptions (e.g., DK2, DK4) are in question, whilst the downwards arrows show whether inconsistencies arise from the newly introduced knowledge, but not how (which will be explained step by step with the theorem prover).

3. Explaining the Rebuttals

We have seen that the `OpenArgue` system allows one to construct security arguments and display these diagrammatically. However, the reasoning service that supplies information on rebuttals relies on detecting inconsistencies through failure to find a model for a formula that captures the logical content of the argument. Currently, we are exploring

the addition of a further reasoning service, the ALLIGATOR theorem prover [8].[6] It will allows us to identify the premises and reasoning that lead to an inconsistency. In this section, we briefly describe the prover, illustrate its use with the PED example, and argue why it is particularly suited for the current task.

3.1. Propositions as Types – Proofs as Objects

The 'propositions as types and proofs as objects' (PTPO) idea underpins the ALLIGATOR theorem prover. It goes back to the work of Curry, Howard and De Bruijn [6,7]. We describe the idea briefly in this subsection

In logic, the semantic notion of a valid argument from premises $P_1,\ldots,P_n$ to a conclusion C, written $P_1,\ldots,P_n \models C$, holds if the truth of the premises carries over to the conclusion. This is complemented by a syntactic notion of derivability of the conclusion from the premises, written $P_1,\ldots,P_n \vdash C$. To trace the reasoning behind an argument, following PTPO, we associate each premise P_x with an object p_x. The notation for this is $p_x : P_x$. The object p_x represents the proof (or evidence) for the premise. Similarly, the conclusion C is associated with an object c. In contrast with the premises, this object is, however, a construction: it is constructed from (some of the) proofs of the premises. Thus, we now write: $p_1 : P_1,\ldots,p_n : P_n \vdash c : C$.

This approach allows us to pose two different questions. Firstly, we can ask the straightforward yes/no-question whether $p_1 : P_1,\ldots,p_n : P_n \vdash c : C$. We are asking whether, given the premises, c is a proof of C. This is a *type checking* problem: given the premises (and the proofs for each of them), is the object c of type C? We treat the proof c as an object and the proposition C as a type, hence the slogan 'proofs as objects, propositions as types'. Secondly, there is the harder question which asks for a proof of the conclusion: $p_1 : P_1,\ldots,p_n : P_n \vdash ?x : C$. In other words, we require a proof object c which, when substituted for the variable $?x$, gives us a valid argument. This is a *theorem proving* problem. A solution, i.e a value for $?x$, represents a proof of C. In fact, c is very compact representation of a natural deduction proof for C. It is beyond the scope of this paper to provide the full set of rules for constructing proof objects for conclusions.[7] The following two examples, however, illustrate the idea.

$$\frac{p : P \in \Gamma}{\Gamma \vdash p : P}\ start \tag{1}$$

This rule says that if the proposition P and its evidence is part of the set of premises (notation Γ), then we can derive P as a conclusion from those same premises, with p as the evidence. Modus ponens is covered by the following rule:

$$\frac{\Gamma \vdash p : P \qquad \Gamma \vdash f : P \to C}{\Gamma \vdash f \cdot p : C}\ arrow - elimination \tag{2}$$

This rule allows us to construct a proof $f \cdot p$ for C from the proof p for P and f for $P \to C$. The constructed proof is obtained through (function) application (signified by '$\cdot$') of the proof for $P \to C$ to the proof for P.

[6] http://mcs.open.ac.uk/pp2464/alligator/

[7] But see the documentation at http://mcs.open.ac.uk/pp2464/alligator/

3.2. The PED example revisited

To adapt the PED example for processing by the theorem prover, we added proof objects for each of the PED propositions. We used mnenonic names, so for instance the proof for a proposition $F1$ becomes $pf1$. Thus we obtained the PED premises Γ_{PED}. We then asked the prover to construct a proof for the contradiction (represented by the symbol $\bot$): $\Gamma_{PED} \vdash ?x : \bot$. On an Intel Core i7 processor the following solution was returned (within less than 0.000 seconds): $pdk4impliesndk2 \cdot pdk4 \cdot pi1(pa9impliesdk2andna4 \cdot (pf10andf11impliesa9 \cdot pair(pf10, pf11)))$. This object can be expanded into the following natural deduction proof:

(3)

$$\dfrac{\dfrac{dk4 \to \neg dk2 \quad dk4}{\neg dk2} \qquad \dfrac{a9 \to (dk2 \wedge \neg a4) \qquad \dfrac{(f10 \wedge f11) \to a9 \qquad \dfrac{f10 \quad f11}{f10 \wedge f11}}{a9}}{dk2 \wedge \neg a4}}{dk2}$$

$$\bot$$

The proof shows that a contradiction arises from proofs for both $dk2$ and $\neg dk2$, where $dk2$ stands for 'Defeating the tamper proofing mechanism costs more than $25,000'. The argument against this proposition (on the left-hand side) is based on $dk4$ – i.e. 'Incentive and expertise (with some imagination) represent enough ingredients to overcome tamper proofing mechanism ' – implying $\neg dk2$. The right-hand side of the tree provides a breakdown of the argument for $dk2$ (which can be explicated in English along similar lines).

For the current application, we have restricted ourselves to the proposition logic. However, the actual prover can also deal with predicate and higher order logics. This makes it possible, at least in principle, to analyse arguments at a finer level of granularity. The proposed proof object-based representations lend themselves particularly well for modeling the content of natural language sentences, including anaphors and presuppositions, which are difficult to model in classical predicate logic (see e.g. Piwek & Krahmer [9], Ranta, [10] and Cooper et al. [2]).

3.3. Arguments with Natural Language Descriptions and Natural Deductions

The combination of visual diagramatic OpenArgue arguments and the Alligator structured proof objects provide a natural way to reason about the security requirements. The naturalness is reflected in two perspectives. First, the natural language descriptions can appear next to the elements in the argument diagram shown in Figure 1. Second, we have seen that the prover delivers a compact representation of a natural deduction proof. Apart from the fact that natural deduction proofs are more suitable for human consumption than, for instance, resolution and tableaux proofs, there are further benefits to the proof object representation. This representation can be checked by an independent short program for correctness, thus satisfying the de Bruijn criterion: A proof assistant satisfies the de Bruijn criterion if it generates proof-objects (of some form) that can be checked by an easy algorithm. [1].

4. Discussions

We have shown, using a concrete example, that a prover with explicit proof objects can supply details on why claims are rebutted, and is therefore a valuable addition to tool support for security arguments. In future work, we aim to fully integrate `OpenArgue` with ALLIGATOR and perform further evaluations. This will include developing assurance arguments for the translation between `OpenArgue` and ALLIGATOR notations and a natural language generation front end to articulate the proof objects.

As an alternative to the traditional PED, recently Apple Pay was announced to be "usable, secure and private"[8]. Although this is a bold new claim, it is yet to be seen whether legal and technical evidence supports it, and whether there are any hidden trust assumptions of which consumers should be made aware. Our contention is that by codifying arguments into a form that can be readily analysed by the argumentation and reasoning tools that are described in this paper will allow one to separate hype from reality.

Acknowledgement Financial support of the ERC (Adv. Grant ASAP No. 291652 project), and SFI (grant 03/CE2/I303_I) are gratefully acknowledged.

References

[1] H. Barendregt and H. Geuvers. Proof-assistants using Dependent Type Systems. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier, 2001.

[2] R. Cooper, S. Dobnik, S. Lappin, and S. Larsson. A Probabilistic Rich Type Theory for Semantic Interpretation. In *Proceedings of the EACL 2014 Workshop on Type Theory and Natural Language Semantics (TTNLS)*, pages 72–79, Gothenburg, Sweden, 2014. Association for Computational Linguistics.

[3] S. Drimer, S. Murdoch, and R. Anderson. Thinking Inside the Box: System-Level Failures of Tamper Proofing . In *SP'2008*, pages 281–295. IEEE Press, May 2008.

[4] V. Franqueira, T. Tun, Y. Yu, R. Wieringa, and B. Nuseibeh. Risk and Argument: A Risk-based Argumentation Method for Practical Security. In *RE'11 Proceedings*, pages 239–248. IEEE Press, 2011.

[5] B. Haley, C. Laney, D. Moffett, and B. Nuseibeh. Using trust assumptions with security requirements. *Requir. Eng.*, 11(2):138–151, Feb. 2006.

[6] W. Howard. The formulae-as-types notion of construction. In J. Seldin and J. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, Inc., Boston, MA, 1980.

[7] R. P. Nederpelt, J. H.Geuvers, and R. C. de Vrijer, editors. *Selected Papers on Automath*. Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1994.

[8] P. Piwek. The ALLIGATOR Theorem Prover for Dependent Type Systems: Description and Proof Sample. In *Proceedings of the Inference in Computational Semantics Workshop (ICoS-5)*, Buxton, UK, 2006.

[9] P. Piwek and E. Krahmer. Presuppositions in Context: Constructing Bridges. In P. Bonzon, M. Cavalcanti, and R. Nossum, editors, *Formal Aspects of Context*, volume 20 of *Applied Logic Series*, pages 85–106. Kluwer Academic Publishers, Dordrecht, 2000.

[10] A. Ranta. Computational semantics in type theory. *Mathematics and Social Sciences*, 165:31–57, 2004.

[11] Y. Yu, T. T. Tun, A. Tedeschi, V. N. L. Franqueira, and B. Nuseibeh. OpenArgue: Supporting Argumentation to Evolve Secure Software Systems. In *RE'11*, pages 351–352. IEEE press, 2011.

[8] http://www.bbc.co.uk/news/technology-29107449